Multi-Action Sampling with Deep Reinforcement Learning for Traveling Salesman Problem

Wei Liu, Thomas Bäck, and Yingjie Fan

LIACS, Leiden University, Leiden, Netherlands

Abstract. The Traveling Salesman Problem (TSP) is a classic combinatorial optimization challenge with broad applications in logistics and transportation. While traditional heuristics remain dominant due to their efficiency and reliability, recent developments in Deep Reinforcement Learning (DRL) have introduced promising new directions for data-driven approaches to solving the TSP. Although DRL-based methods show promise, they often do not yet match classical heuristics in terms of computational efficiency and solution quality. One promising direction to bridging this gap is the integration of classical heuristics with learning-based methods. The Learn-to-Improve (L2I) framework follows this hybrid paradigm, combining heuristics with reinforcement learning to iteratively refine solutions. In this paper, we propose a novel multi-action sampling strategy that further enhances the L2I framework for solving TSP. The core idea is to improve solution quality by averaging rewards over multiple actions, which reduces bias and encourages more effective exploration compared to single-action strategies. During inference, multi-action sampling is applied in the later stages to explore diverse solution paths in parallel, helping to prevent premature convergence. Experimental results demonstrate that the proposed method outperforms existing L2I approaches while maintaining competitive computational efficiency.

Keywords: Vehicle Routing · Deep Reinforcement Learning · Multi-action Sampling · Neural Networks · Learning-based Optimization

1 Introduction

Vehicle Routing Problems (VRPs) are critical yet challenging tasks in transportation and logistics due to their NP-hard nature. Efficient solution methods are essential to enable scalable and effective routing systems. Among these problems, the Traveling Salesman Problem (TSP) is a foundational case, with extensive research and wide-ranging applications in logistics, transportation, and scheduling domains [1]. Efficiently solving the TSP is a practical step toward addressing more complex VRPs.

Over the past few decades, numerous algorithms have been developed to tackle the TSP, enhancing both the quality of the solution and computational efficiency. Traditional approaches, including exact algorithms and heuristics algorithms [2], have been extensively studied and applied. Exact methods, such as

branch-and-bound [3], ensure optimal solutions but are often limited by scalability due to their exponential time complexity. Moreover, heuristic algorithms [4][5][6], while capable of producing promising solutions in significantly less time, are more practical for large-scale routing problems. However, these methods often rely on manually designed decision rules that require extensive domain knowledge, which can limit their adaptability and performance on unfamiliar or complex problem instances.

The development of machine learning, particularly deep reinforcement learning (DRL) [7], has introduced promising alternatives for solving routing problems. DRL-based methods can outperform traditional heuristics by offering faster inference and removing the need for manually crafted rules. These models are more generalizable and adaptive, as they learn decision-making policies through environment interaction. DRL for VRPs can be categorized into two main approaches: Learn-to-Construct (L2C) and Learn-to-Improve (L2I) [8]. In L2C, the trained model represents an approximate implicit mapping function between the input and the final solution. During inference, solutions are generated sequentially by predicting one decision at a time, incrementally constructing a complete solution by adding a node to a partial solution at each step. In contrast, L2I uses trained models as auxiliary components for improvement operators. For instance, they can be used to select node pairs for local search operators or to guide destroy-repair operators. Unlike L2C approaches, which construct solutions from scratch, L2I methods start with an initial complete solution and iteratively refine it to enhance the quality of solutions.

In this paper, we propose a novel training and inference strategy to enhance L2I methods for solving the TSP. Unlike existing L2I approaches that rely on fixed or heuristic-based improvement operations, limiting their adaptability. Our method introduces multi-action sampling in both training and inference phases to support broader exploration and more effective learning. During training, the multi-action sampling method facilitates more stable policy training by averaging the rewards of multiple sampled actions. In addition, we employ multi-action parallel sampling method during the later stage of inference to systematically explore the unknown region of the optimal solutions. Thus, our proposed method enables better exploration of the solution space and reduces the risk of premature convergence.

Extensive experimental evaluations demonstrate that our proposed method improves solution quality and training stability, delivering superior performance compared to existing L2I baselines while maintaining competitive computational efficiency. It is important to note that the objective of this paper is not to develop a learning-based optimization method that competes directly with exact solvers such as Concorde, or classical heuristics like evolutionary algorithms. Rather, our goal is to investigate the extent to which learning-based methods can perform in terms of computational efficiency and solution quality within their own paradigm. Accordingly, we primarily compare our approach against other learning-based methods. To provide a clearer picture of the optimality gap

between learning-based and classical approaches, we also include results from representative classical heuristics in our experimental evaluation.

2 Related Work

Bello et al. [9] introduced a DRL-based Pointer Network that replaced manually designed local search heuristics and outperformed its supervised counterpart on TSP50 and TSP100. However, it was not suitable for the capacitated vehicle routing problem (CVRP). Nazari et al. [10] extended this approach by incorporating dynamic embeddings, enabling adaptation to evolving customer demands and reducing the inference time by 60% compared to OR-Tools [13].

With the development of Graph Neural Networks (GNNs) [12], researchers observed that Pointer Networks struggle to effectively capture the complex topological structures in routing problems. As a result, instead of using the sequence-to-sequence paradigm, Khalil et al. [11] tackled the TSP by integrating reinforcement learning with a GNN-based approach called Structure2Vec. Their model achieved performance comparable to the model in [9].

Inspired by the success of Transformers [14] in sequential decision problems, Deudon et al. [15] were the first to apply the transformer architecture, using multi-head attention and a feed-forward encoder-decoder with a pointing mechanism in the decoding process to solve the TSP. Joshi et al. [16] introduced a graph convolutional network (GCN) trained under a supervised learning framework for TSP, while Kool et al. [17] retained the multi-head attention-based encoder and incorporated self-attention along with a modified context node vector during decoding. Their work presented a general framework for solving VRPs and remains one of the most widely adopted end-to-end approaches.

Ma et al. [18] proposed graph pointer networks with DRL for TSP, achieving better generalization on instances up to 1,000 nodes than previous learning-based methods. Peng et al. [19] built on [17] by introducing a dynamic attention model with an adaptive encoder-decoder architecture for CVRP, effectively incorporating graph structure information throughout the optimization process.

Kwon et al. [20] addressed symmetry issues in DRL-based TSP solutions by introducing parallel rollouts using the POMO framwork. In addition, they applied data augmentation to increase training diversity and overcome the limited number of solution trajectories.

Xin et al. [21] improved the model from [17] by using multiple decoders with distinct parameters, selecting the best decoder output to determine the next node. Fu et al. [22] successfully generalized a small pre-trained model to large TSP instances. Jiang et al. [23] exploited group distributionally robust optimization and designed a convolutional embedding layer to improve the generalization of solutions cross different data distributions.

Chen and Tian [24] proposed the first DRL-based L2I model, selecting solution fragments for re-optimization through a dual-policy framework. Lu et al. [25] enhanced this by incorporating a richer set of improvement operators and

a threshold-based mechanism to decide between further refinement or perturbation.

Wu et al. [27] introduced Learning Improvement Heuristic (LIH), a DRL-based model that leverages a Transformer-like architecture to select node pairs for 2-opt operations in TSP and CVRP, which Ma et al. [26] later improved by incorporating cyclic positional encoding to better capture solution symmetry.

Hottung and Tierney [28] applied DRL to Large Neighborhood Search (LNS) for CVRP, training a pointer network to guide repair operations while relying on manually designed heuristics. Gao et al. [29] addressed this limitation by developing EGATE, a graph-attention network that used edge embeddings to enhance neighborhood search efficiency.

Xin et al. [30] integrated DRL-based learning with the powerful LKH heuristic, training a sparse graph network to guide edge selection, significantly improving TSP scalability. Li et al. [32] developed a divide-and-conquer strategy for large-scale CVRP, using a Transformer model to predict sub-problem costs before solving and merging them, achieving an order of magnitude speedup over LKH3 [31] while maintaining solution quality.

L2I methods iteratively refine complete solutions, gradually improving their quality in multiple steps. Although these approaches are effective in finding high-quality solutions, they typically entail higher computational costs during both training and inference compared to L2C methods, as they require exploring a significantly larger search space.

3 Preliminaries

3.1 Traveling Salesman Problem

The goal of the Traveling Salesman Problem (TSP) is to determine the shortest possible route that allows a salesman to start and end at the same city while visit each city in a given set exactly once. Let $C = \{c_1, c_2, \ldots, c_n\}$ denote the set of cities, where n is the total number of cities. The distance between any two cities i and j is denoted by $d(i, j)$. The objective is to find a permutation $x = (x_1, x_2, \ldots, x_n)$ of the cities that minimizes the total length of the tour. This can be formulated as the following minimization problem:

$$f_{\text{TSP}}(x) = \sum_{i=1}^{n-1} d(c_{x_i}, c_{x_{i+1}}) + d(c_{x_n}, c_{x_1}). \tag{1}$$

This expression represents the total distance traveled by visiting the cities in the order defined by x, including the final return to the starting city.

3.2 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) differs from traditional Deep Learning (DL) by learning decision-making policies through interactions with dynamic environments. At each time step t, an agent observes the current state s_t of the

environment and selects an action a_t. The environment then transitions to a new state s_{t+1} and provides the agent with a reward r_t as feedback. This iterative process enables the agent to learn optimal strategies without the need for labeled data, an inherent limitation in supervised learning (SL).

The ability of DRL to explore and interact with the environment grants it superior generalization capabilities compared to SL methods. Therefore, DRL has been extensively applied to optimization problems in routing and logistics.

To formalize this, we define the agent's objective as maximizing the expected return (i.e., negative tour length) under policy p_θ for a given instance S as:

$$J(\theta \mid S) = -E_{\tau \sim p_\theta(\cdot|S)}[L(\tau \mid S)], \tag{2}$$

where S represents a VRP instance, τ is a solution trajectory, and $L(\tau \mid S)$ is the tour length (negative reward). The policy $p_\theta(\tau \mid S)$ denotes the probability distribution of selecting solution τ based on parameters θ.

To optimize θ, the REINFORCE algorithm is commonly employed. Its gradient is given by:

$$\nabla_\theta J(\theta \mid S) = E_{\tau \sim p_\theta(\cdot|S)} \left[(L(\tau \mid S) - b(S)) \cdot \nabla_\theta \log p_\theta(\tau \mid S) \right], \tag{3}$$

where $b(S)$ is a baseline function to reduce variance and stabilize training.

Recent advancements, such as the POMO method [20], have further improved the efficiency of DRL in VRPs by incorporating shared baselines and multiple starting points, enhancing both solution quality and training stability.

4 Methodology

4.1 Multi-Action Sampling in Training

Each TSP instance consists of a set of N nodes, where each node i is represented by its 2D coordinates and other problem-specific features. In conventional DRL approaches, the positional features and coordinates of the nodes are fed into a neural network as input. The policy network then generates a probability matrix, from which a single action is selected at each time step based on the calculated probabilities. The selected action is subsequently applied to modify the original solution through state transitions in the environment. The path length $f(s_{t+1})$ of the updated state s_{t+1} is used as a reward to update the network, as illustrated in the left part of Fig. 1.

In this paper, we extend the standard framework by performing multiple action selections at each time step (see MS-improve in Fig. 1), enhancing exploration and improving solution quality. The policy network follows an encoder-decoder architecture with multi-layer self-attention, aligning with the design proposed in [27]. This architecture, inspired by Transformer models [14], enables the network to capture spatial dependencies among nodes by encoding the problem instance into contextual representations and decoding actions based on learned attention weights.

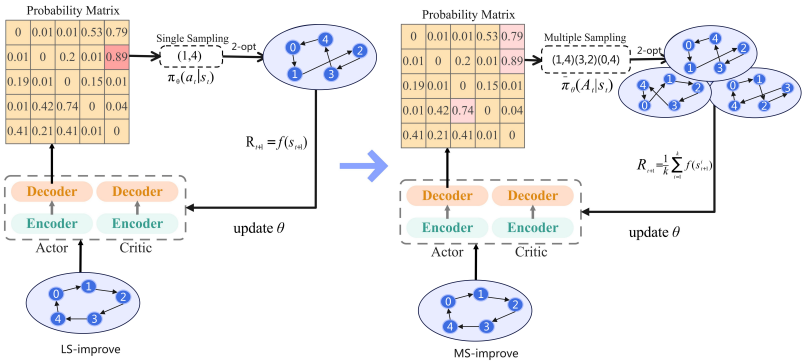

Fig. 1. Actor-Critic Learning Framework for TSP with 2-Opt Improvement Heuristic.

During the training phase, we propose a *multi-action sampling strategy* to improve the efficiency of policy learning. Instead of selecting a single action per time step from the policy network, we perform multiple samplings based on the probability distribution to obtain multiple node pairs. For each sampled action, we compute the corresponding solution length and use the average of these lengths to calculate the reward. In addition, we update the policy network using the average probability of the selected actions.

Action Set Generation: The policy network encodes the solution information using a neural network (encoder-decoder) to generate a probability matrix for node pairs. At time step t, a set of k actions $A_t = \{a_t^1, a_t^2, \ldots, a_t^k\}$ is sampled from this matrix. The 2-opt heuristic is then applied to each selected node pair to modify the original path, resulting in a new state and an improved solution.

Reward Computation: For each action a_t^i in the sampled action set, the corresponding new state s_{t+1}^i is obtained by applying the 2-opt operator. The solution length $f(s_{t+1}^i)$ is then computed. The reward at step t is defined as the average tour length across the k generated solutions:

$$R_t = \frac{1}{k} \sum_{i=1}^{k} f(s_{t+1}^i). \tag{4}$$

This reward averaging reduce the variance introduced by individual actions, providing a more stable learning process and encouraging more balanced policy updates.

Policy Update: To update the policy network, we compute the average probability of the sampled actions in the action set A_t:

$$\bar{\pi}_\theta(A_t|s_t) = \frac{1}{k} \sum_{i=1}^{k} \pi_\theta(a_t^i|s_t). \tag{5}$$

The policy parameters are then updated using the REINFORCE gradient estimator:

$$\nabla_\theta J(\theta) = E\left[\sum_{t=0}^{T-1} \nabla_\theta \log \bar{\pi}_\theta(A_t|s_t) G_t\right], \tag{6}$$

where G_t represents the cumulative reward from time step t onward. By using the average probability of multiple sampled actions, this update strategy ensures stable policy learning and prevents fluctuations caused by high-probability or high-reward individual actions.

The proposed multi-action sampling strategy enhances exploration by evaluating multiple actions simultaneously, thereby reducing the risk of premature convergence to suboptimal solutions. It mitigates policy learning bias by leveraging averaged rewards and log-probabilities, ensuring a more balanced update process. In addition, it improves generalization by learning from a wider variety of solution trajectories, enhancing adaptability across various problem instances. Lastly, it improves the stability of training by smoothing the gradient updates, reducing fluctuations, and facilitating a more stable learning process.

4.2 Multi-Action Sampling in Inference

We also employ the multi-action sampling strategy during the inference stage to enhance solution exploration. After training, the model performs inference based on the learned policy, iteratively refining the initial path through a sequence of 2-opt operations at each time step until reaching the predefined time step threshold. Here, we introduce a multi-action sampling mechanism in the later stages of inference to expand the search space. Specifically, at each time step n, we sample k candidate actions from the probability distribution, applying k distinct 2-opt transformations to the solutions obtained at step $n-1$. This process generates k candidate paths, from which we select the shortest as the updated solution for the next time step. These k transformations are executed in parallel in our experiment with minimal computational overhead, ensuring efficient search expansion.

5 Experiments

In this section, we evaluate the performance of the proposed multi-action sampling approach on the TSP. We generate synthetic datasets for TSP with varying numbers of nodes (20, 50, and 100). The coordinates of the nodes were sampled uniformly from the unit square [0,1]x[0,1]. All experiments were carried out on a system with an Intel i7-9700K CPU, 32 GB RAM, and an NVIDIA RTX 3090 GPU. The code was implemented in Python using Pytoch 2.3.0. The policy and value networks were trained using the Adam optimizer with a learning rate of 10^{-4}. We evaluate our method with a variety of baselines, including: 1) concorde [33], an efficient exact solver specialized for TSP; 2) LKH3 [31], a well-known heuristic solver that achieves state-of-the-art performance on various routing

problems; 3) OR-Tools [13], a mature and widely used solver for routing problems based on metaheuristics; and 4) AM [17], the pioneering application of the transformer model to solving VRPs, which has yielded outstanding results. 5) GNNGLS [34], a hybrid data-driven approach based on graph neural networks and guided local search to solve the TSP. 6) LS-Improve [27], a classic learning-to-improve technique that iteratively applies 2-opt swaps to refine the initial solution, ultimately achieving an optimal route.

5.1 Results and Discussion

Table 1. Performance comparison across various methods for TSP20, TSP50, and TSP100. The table presents the objective value (Obj.), optimality gap (Gap), and computation time (Time) for each method. *MS-Improve (RO)* refers to our method with the modified reward mechanism only, while *MS-Improve (RO + MA20)* and *MS-Improve (RO + MA50)* incorporate both the modified reward mechanism and multi-action sampling with 20 and 50 actions, respectively, during inference.

Methods	TSP20			TSP50			TSP100		
	Obj.	Gap	Time	Obj.	Gap	Time	Obj.	Gap	Time
Concorde	3.83	0.00%	<1m	5.69	0.00%	2m	7.76	0.00%	20m
LKH3	3.83	0.00%	<1m	5.69	0.00%	1.5m	7.76	0.00%	14m
OR-Tools	3.86	0.94%	1.2m	5.85	2.87%	5m	8.06	3.86%	23m
AM (N=1,280)	3.84	0.30%	1.8m	5.72	0.48%	7.4m	7.95	2.45%	1.7h
AM (N=5,000)	3.83	0.04%	16m	5.72	0.47%	45m	7.93	2.18%	4.3h
GNNGLS	3.83	0.00%	-	5.69	0.01%	-	7.81	0.70%	-
LS-Improve ($T = 3,000$)	3.83	0.00%	39m	5.71	0.34%	45m	7.91	1.85%	1.5h
MS-Improve (RO)	3.83	0.00%	39m	5.70	0.20%	45m	7.85	1.10%	1.5h
MS-Improve (RO + MA20)	3.83	0.00%	42m	5.69	0.00%	1.1h	7.81	0.64%	1.7h
MS-Improve (RO + MA50)	3.83	0.00%	47m	5.69	0.00%	1.3h	7.78	0.24%	2.2h

We evaluate all algorithms on 10,000 randomly generated TSP instances and report their average solution quality and runtime. For exact solvers such as Concorde and LKH3, the runtime per instance is extremely short. Since we batch 10,000 instances during evaluation, the reported total runtimes are approximate and may slightly overestimate due to I/O and orchestration overhead. The comparative performance of various methods across TSP20, TSP50, and TSP100 is summarized in Table 1.

Our proposed MS-Improve algorithm demonstrates strong performance, particularly when multi-action sampling is used during inference, achieving near-optimal solutions across all problem sizes. For TSP20, we successfully achieved the optimal objective value 3.83 even without multi-action sampling during inference. For TSP50 and TSP100, our enhanced reward mechanism yielded superior solutions compared to the LS-Improve method. Furthermore, the implementation of multiple sampling techniques during the inference phase resulted in additional performance improvements.

Notably, our approach (*RO + MA50*) achieves the best performance among learning-based methods, attaining an objective value of 7.78 with a minimal optimality gap of 0.24%. While the computation time of *RO + MA50* (2.2 hours) exceeds that of LKH3 (23 minutes), it remains competitive with other learning-based methods such as LS-Improve (1.5 hours).

Although our algorithm increases the execution time compared to the original LS-Improve, it supports parallel execution of path exchanges. In other words, the training time required to perform 20 actions (samples) is almost equivalent to that for 50 actions (samples), excluding the time spent sequentially selecting actions.

In summary, our algorithm, MS-Improve, achieves near-optimal solutions with competitive computational time, demonstrating its effectiveness and scalability for solving TSP across varying instance sizes.

5.2 Ablation Study

To better understand the contributions of individual components in MS-Improve, we perform a series of ablation studies. Specifically, we analyze the effects of the proposed multi-action reward computation during training and the multi-action sampling strategy during inference.

Impact of Multi-Action Reward Computation on Training

To examine the effect of the proposed multi-action reward computation method on the training process, we compare the training dynamics of LS-Improve with those of our method. As shown in Fig. 2, the multi-action reward computation significantly improves training stability compared to LS-Improve. Specifically, the objective value curves for our method exhibit notably smaller fluctuations, indicating a more consistent and stable convergence trajectory.

Moreover, the proposed method effectively accelerates the reduction of the objective value during early epochs, achieving competitive performance within fewer iterations. This improvement is primarily attributed to the averaging of rewards across multiple sampled actions, which reduces the variance in the policy gradient updates and improve learning efficiency.

Impact of Multi-Action Sampling on Inference

We further investigate the impact of the number of sampled actions on inference performance by conducting a controlled study on multi-action sampling.

Fig.3 illustrates the performance variations across different values of k and corresponding inference time steps. Here, the time step represents the point at which multi-action sampling is activated, and k denotes the number of actions sampled per step. Notably, $k = 1$ serves as the baseline, where only one action is selected at each time step to update the solution.

The results demonstrate that applying larger k values at earlier time steps (500–600) may cause instability in the convergence process, resulting in higher

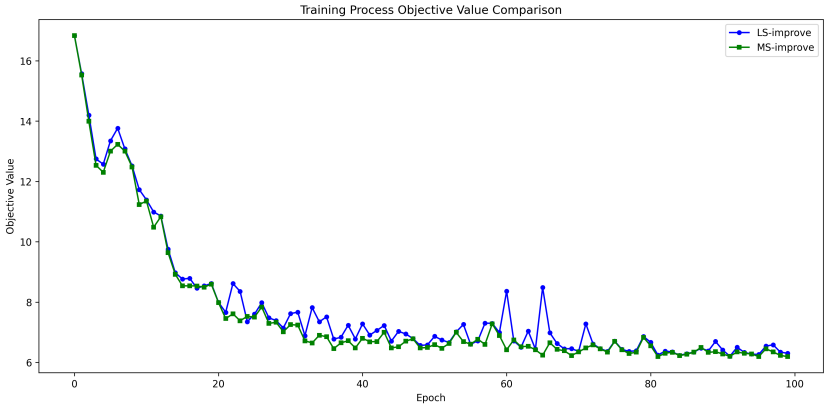

Fig. 2. Performance Variations Across Different Reward Computation Methods in the Training Phase of TSP50.

objective values. In contrast, introducing multi-action sampling at later time steps (900–1000) significantly improves solution quality. During these later stages, larger k values exhibit clear advantages by achieving lower objective values and further optimizing path quality. This suggests that while fewer actions are preferable in the early inference phase to maintain stability, leveraging a greater number of actions in the later phases expands the search space for potential optimal solutions, thereby improving the overall performance.

For instance, at the 500 time step, sampling fewer actions ($k = 20$) results in better objective values compared to higher k values, such as $k = 40$ or $k = 50$. However, as the time step progresses to 1000, the performance of larger k values overtakes that of smaller ones, indicating that multi-action sampling at later stages effectively enhances exploration and solution quality. This analysis underscores the importance of adapting the sampling strategy to different phases of inference, balancing exploration and convergence to achieve optimal results.

6 Conclusion

In this paper, we proposed a novel multi-action sampling strategy to enhance the the training efficiency of Learn-to-Improve (L2I) methods for VRPs. Our approach integrates multi-action sampling in both the training and inference stages. By averaging rewards over multiple sampled actions during training, our method mitigates bias and stabilizes policy updates. In inference, we employ multi-action sampling to explore more possibilities of optimal solutions in parallel, reducing the risk of premature convergence and enhancing final solution quality.

Extensive experiments on the TSP demonstrate the effectiveness of our approach. Compared to state-of-the-art DRL baselines, our method achieves superior or comparable solution quality while maintaining competitive computa-

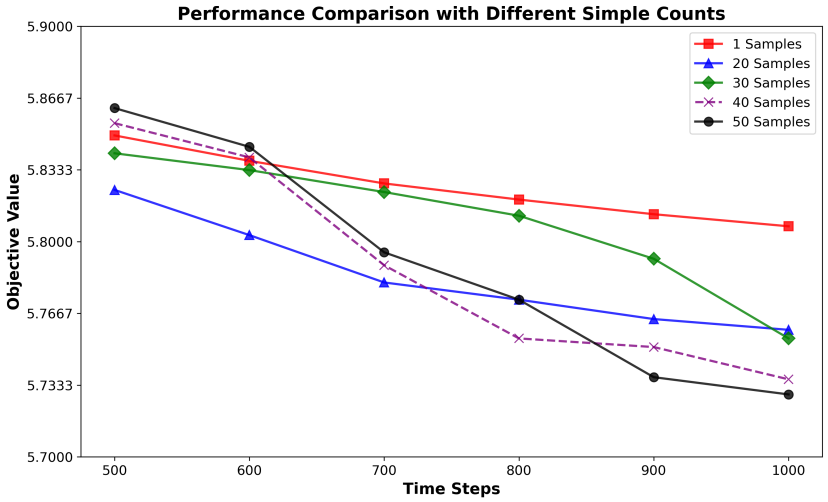

Fig. 3. Performance Variations Across Different k Values and Time Steps in the Inference Phase of TSP50.

tional efficiency. The ablation study further validates the impact of multi-action sampling, showing that increasing the number of sampled actions improves optimization performance without significantly increasing computational overhead. From this perspective, dynamically adjusting the number of actions during the inference stage may further improve both solution quality and inference efficiency.

As a next step, we aim to further explore the potential of combining classical heuristics with learning-based methods, with the goal of developing more efficient optimization approaches for large-scale and real-world problems that are difficult to model explicitly.

References

1. G. Laporte, A. Asef-Vaziri, and C. Sriskandarajah, "Some applications of the generalized travelling salesman problem," *Journal of the Operational Research Society*, vol. 47, no. 12, pp. 1461–1467, 1996.
2. P. Festa, "A brief introduction to exact, approximation, and heuristic algorithms for solving hard combinatorial optimization problems," in *Proceedings of the 16th International Conference on Transparent Optical Networks*, 2014.
3. E. L. Lawler and D. E. Wood, "Branch-and-bound methods: A survey," *Operations Research*, vol. 14, no. 4, pp. 699–719, 1966.
4. N. Ghanem, C. Altehoefer, A. Furtwängler, J. Winterer, O. Schäfer, O. Springer, E. Kotter, and M. Langer, "Computed tomography in gastrointestinal stromal tumors," *European Radiology*, vol. 13, pp. 1669–1678, 2003.
5. X. Xiang, Y. Tian, J. Xiao, and X. Zhang, "A clustering-based surrogate-assisted multiobjective evolutionary algorithm for shelter location under uncertainty of road

networks," *IEEE Transactions on Industrial Informatics*, vol. 16, no. 12, pp. 7544–7555, 2020.
6. Y. Su, N. Guo, Y. Tian, and X. Zhang, "A non-revisiting genetic algorithm based on a novel binary space partition tree," *Information Sciences*, vol. 512, pp. 661–674, 2020.
7. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, and D. Hassabis, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
8. R. Shi and L. Niu, "A brief survey on learning based methods for vehicle routing problems," 2025.
9. Bello, I., Pham, H., Le, Q. V., Norouzi, M., Bengio, S.: Neural combinatorial optimization with reinforcement learning. In: International Conference on Learning Representations (ICLR), 2017.
10. Nazari, M., Oroojlooy, A., Snyder, L., Takác, M.: Reinforcement learning for solving the vehicle routing problem. Advances in Neural Information Processing Systems, vol. 31 (2018).
11. Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L.: Learning combinatorial optimization algorithms over graphs. Advances in Neural Information Processing Systems, vol. 30 (2017).
12. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, S. P.: A comprehensive survey on graph neural networks. IEEE Transactions on Neural Networks and Learning Systems, 32(1), 4–24 (2020).
13. Google Inc.: OR-Tools: Google optimization tools. Available at: https://developers.google.com/optimization/.
14. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. In: Proceedings of Neural Information Processing Systems, pp. 6000–6010 (2017).
15. Deudon, M., Cournut, P., Lacoste, A., Adulyasak, Y., Rousseau, L.-M.: Learning heuristics for the TSP by policy gradient. In: Integration of Constraint Programming, Artificial Intelligence, and Operations Research, pp. 170–181 (2018).
16. Joshi, C. K., Laurent, T., Bresson, X.: An efficient graph convolutional network technique for the Traveling Salesman Problem. arXiv preprint arXiv:1906.01227.
17. Kool, W., Van Hoof, H., Welling, M.: Attention, learn to solve routing problems! In: International Conference on Learning Representations (ICLR), 2019.
18. Ma, Q., Ge, S., He, D., Thaker, D., Drori, I.: Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. In: AAAI Workshop on Deep Learning on Graphs: Methodologies and Applications, 2020.
19. Peng, B., Wang, J., Zhang, Z.: A deep reinforcement learning algorithm using dynamic attention model for vehicle routing problems. In: Artificial Intelligence Algorithms and Applications, pp. 636–650 (2020).
20. Kwon, Y., Choo, J., Kim, B., Yoon, I., Gwon, Y., Min, S.: POMO: Policy optimization with multiple optima for reinforcement learning. Advances in Neural Information Processing Systems, vol. 33, pp. 21188–21198 (2020).
21. Xin, L., Song, W., Cao, Z., Zhang, J.: Multi-decoder attention model with embedding glimpse for solving vehicle routing problems. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 12042–12049 (2021).
22. Fu, Z., Qiu, K., Zha, H.: Generalize a small pre-trained model to arbitrarily large TSP instances. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 35, pp. 7474–7482 (2021).

23. Jiang, Y., Wu, Y., Cao, Z., Zhang, J.: Learning to solve routing problems via distributionally robust optimization. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 36, pp. 9786–9794 (2022).
24. Chen, X., Tian, Y.: Learning to perform local rewriting for combinatorial optimization. In: Proceedings of Neural Information Processing Systems, pp. 6278–6289 (2019).
25. Lu, H., Zhang, X., Yang, S.: A learning-based iterative method for solving vehicle routing problems. In: International Conference on Learning Representations (ICLR), 2020.
26. Ma, Y., Li, J., Cao, Z., Song, W., Zhang, L., Chen, Z., Tang, J.: Learning to iteratively solve routing problems with dual-aspect collaborative transformer. Advances in Neural Information Processing Systems, vol. 34, pp. 11096–11107 (2021).
27. Wu, Y., Song, W., Cao, Z., Zhang, J., Lim, A.: Learning improvement heuristics for solving routing problems. IEEE Transactions on Neural Networks and Learning Systems, pp. 1–13 (2021).
28. Hottung, A., Tierney, K.: Neural large neighborhood search for the capacitated vehicle routing problem. In: ECAI 2020, pp. 443–450 (2020).
29. Gao, L., Chen, M., Chen, Q., Luo, G., Zhu, N., Liu, Z.: Learn to design the heuristics for vehicle routing problem. arXiv preprint arXiv:2002.08539.
30. Xin, L., Song, W., Cao, Z., Zhang, J.: Neurolkh: Combining deep learning model with Lin-Kernighan-Helsgaun heuristic for solving the traveling salesman problem. Advances in Neural Information Processing Systems, vol. 34, pp. 7472–7483 (2021).
31. Helsgaun, K.: An effective implementation of the Lin–Kernighan traveling salesman heuristic. European Journal of Operational Research, vol. 126, no. 1, pp. 106–130 (2000).
32. Li, S., Yan, Z., Wu, C.: Learning to delegate for large-scale vehicle routing. Advances in Neural Information Processing Systems, vol. 34, pp. 26198–26211 (2021).
33. D. Applegate, R. Bixby, V. Chvátal, and W. Cook, "Concorde TSP solver," Available at http://www.math.uwaterloo.ca/tsp/concorde, 2006.
34. B. Hudson, Q. Li, M. Malencia, and A. Prorok, "Graph neural network guided local search for the traveling salesperson problem," in *Proceedings of the International Conference on Learning Representations*, 2021.